Migrating RTOS Software to HPC environment Direct code porting may not be feasible

COVESA AMM 2024 Fall

Hisao Munakata

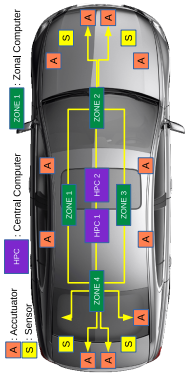
Renesas Electronics Corp.
High Performance Computing Product Group

2024-9-25

Vehicle Computer (HPC) is a one of key element of SDV

Vehicle Computer (HPC) requires Linux + Hypervisor

- To manage unique characteristics of HPC
  - massive parallel execution (x8, x16, x32, x64 cores)
  - Dynamic resource allocation and share
  - Virtualization support
  - Cloud resource utilization (micro-service aware)
- Still needs traditional automotive requirements
  - Deadline schedule management (realtime)
  - 00red0Fast boot0000, Error recovery, Diagnosis
  - Functional safety (ASIL-B/D compliance)



People expect HPC will consolidates multiple existing ECUs roles

RTOS and Linux/HV differences

Difference between ”general” and ”embedded” computing

General purpose computer

- Examples = PC, Smartphone
- User defines machine-usage by installing necessary application
- Need to run on various HWs
- Enough compute resources（to support various use-case）
- GUI with mouse/key operations
- HW can expand

Embedded device

- Home appliances, industry, game
- Predefined usage（single-purpose）
- Tough price competition due to all vendors aims the identical target
- Minimal compute resources
- SW freeze at the sales (static)
- Use dedicated physical button
- No HW expandability

Embedded SW need to work under the heavily restricted condition

Linux vs. RTOS: How its management principle differs

Linux manages dynamic coordination

- A general-purpose compute
  - super computers (science, medical)
  - servers (finance, securities)
  - vehicles and smartphones
  - IoT、Router
- Multi-user, Multi-task
- Memory protection, Virtualization
- Open Source（+30 years）

RTOS represents complete harmonization

- Machine control (embedded)
  - White goods
  - Automotive ECU
- For MCU（In most case single-core）
- Provides real-time guarantees
- Supports functional safety
- OSS and Commercial products
  - OSS：FreeRTOS、Zephyr, etc
  - Commercial：SafeRTOS、QNX, etc

Task control policy of RTOS is completely differ from Linux's policy

Embedded SW aims to achieve perfectly pre-harmonized world

Perfect pre-harmonization

- Deterministic task execution order → task priority assignment
- Interrupt handling → manages interrupt-disabled period
- Memory resource management → static memory allocation
- Hardware resources are statically assigned, meaning one application dominates the resources
- In embedded system SW designer must realizes fully perceive entire system behavior, and fully responsible to implement error recovery algorithm for any kind of failure case to secure perfect harmonization.

RTOS help realizing complete harmony under resource constraints

Chasing both robustness and flexibility

SW design methodology: two major paradigm

HW First (traditional embedded)

- HW spec is fixed upfront
- Must fully utilize HW performance
- Do not care for SW-reuse
- Dedicated dev-environment
  - emulator, debugger
  - Use custom libraries
  - BSP ← Board specific code
- Must create fully pre-harmonized world、furthermore automotive requires functional safety

SW First (smartphone, etc)

- Designed to run on various HW
- Wider HW adaptability
  - processor power
  - memory, storage
  - display size, aspect
- App. development go in parallel
- HW and SW are decoupled
- Fully rely on high-performance OS coordination capability for execution order management

Smart-phone rely on OS built-in auto-coordination capabilities

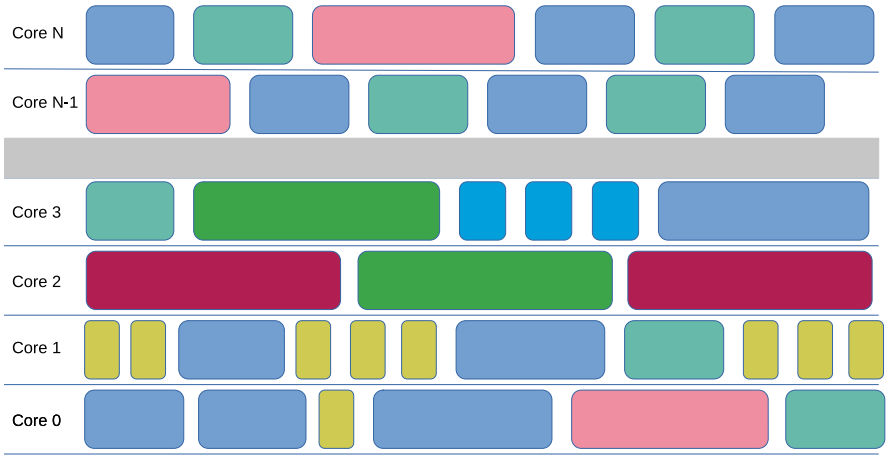
Smart-phone is general computer other than single-purpose (embedded)

- Application choice by user and timely SW update is the essence of Smart-phone, and intentionally give up pre-harmonization by the SW designer. Alternatively, smart-phone realize SW first world by relying on "OS built-in auto-coordination."
- "Auto-coordination feature", which is integrated in the general purpose OS
  - Smart task scheduling
  - Interrupt handling optimization
  - Speculative cache use（Fully utilize free RAM area as a soft cache)
  - Resource allocation, cross-process share (memory, HW IP)

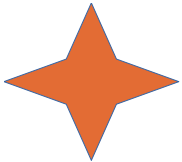
Can we rely on "OS auto-coordination" for entire SDV programs ?

Port existing RTOS code to the HPC

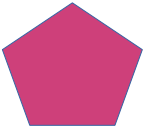
Linux program execution = dynamic execution coordination



RTOS program: self-contained and stand-alone



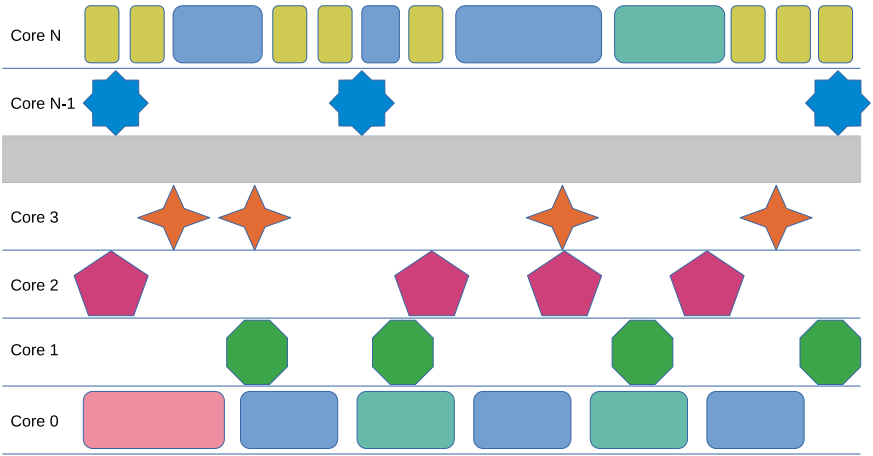
- Each RTOS programs are locally fully optimized (pre-harmonized)
  - Static memory allocation
  - Strict sequence design (real-time)
  - Local interface (not compatible)



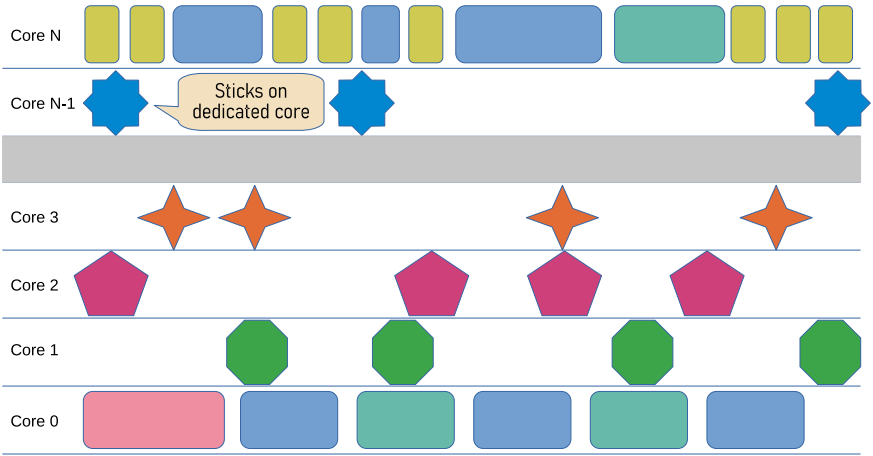
- However, not designed for
  - Compatible API use
  - Shared design
  - System-wide optimization

people do not want to touch already stabilized RTOS programs

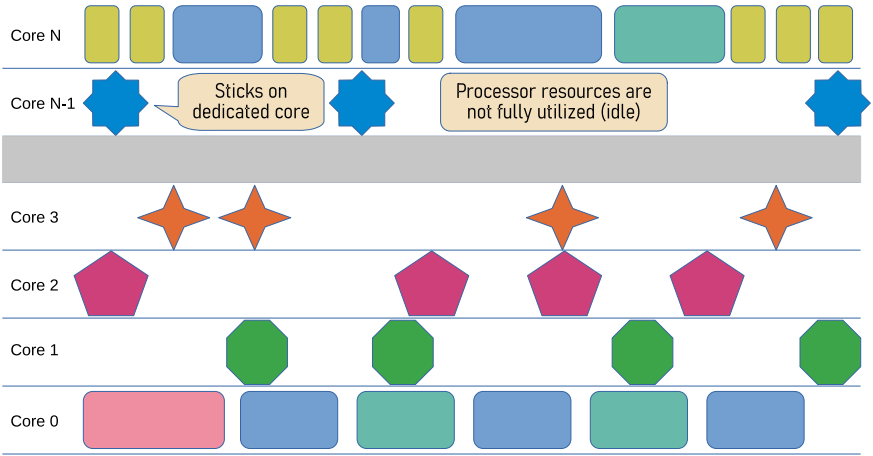
Straight porting of RTOS code to HPC likely fails



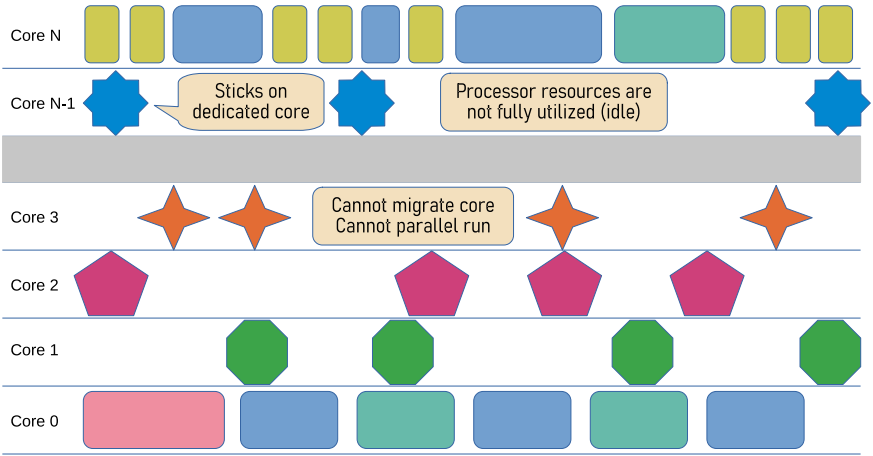
Straight porting of RTOS code to HPC likely fails



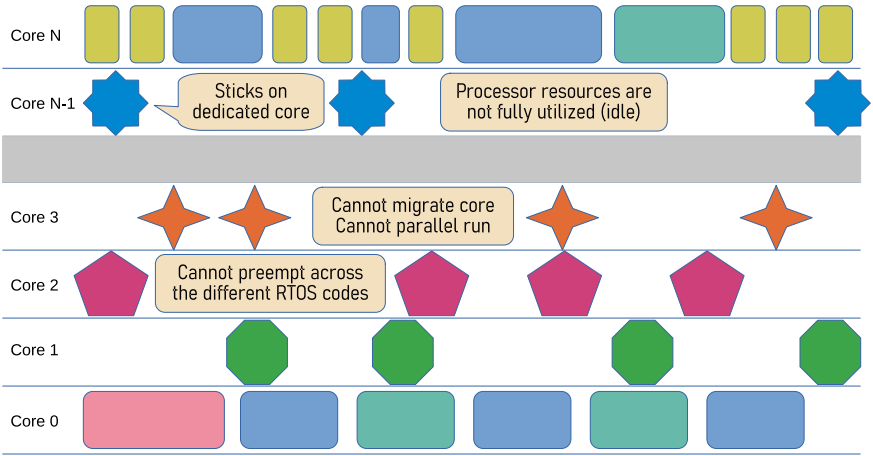
Straight porting of RTOS code to HPC likely fails



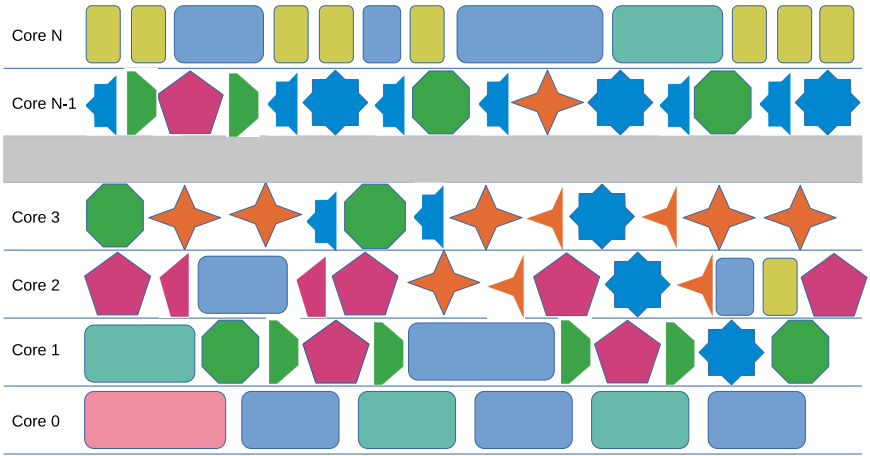
Straight porting of RTOS code to HPC likely fails



Straight porting of RTOS code to HPC likely fails



Ideal RTOS code porting to HPC (but, it not straightforward)



Best SW porting practice to utilize HPC concept

Parallelism rather than processing density is essential for multi-core HPC

- Investigate what RTOS SW manage further
  - Check deadline conditions (Do not straight porting)
  - Any chance to free CPU in code sequences
- Minimize the synchronization (wait) to realize stateless code structure
- Fully utilize OS's auto-coordination capabilities
  - Minimize the use of REALTIME task
  - Optimize kernel scheduling parameter
  - Offload timing constraint process to co-processor
  - Allocate enough memory resource
- HW partitioning by using hypervisor
- Application encapsulation using container

Summary

- In embedded systems for specific applications, designers use RTOS' to create a perfect scheduling world view under the constraint of minimum resources required for cost competition.
- Since perfect scheduling is not possible in PCs and smartphones, the dynamic auto-adjustment mechanism of a high-performance OS is used to run the system on the available resources.
- SDV is able to mount a high-performance OS capable of dynamic auto-scheduling on a vehicle compute (HPC) that has sufficient capacity. The SW part can be updated after the product is shipped to provide the user with the most advanced user experience (value).
- If the SW of RTOS with perfect scheduling is ported to HPC as it is (i.e., if Linux is configured like RTOS), there will be various adverse effects.